linaro

Give me back my GPIO persistence! (Introducing the gpio-manager)

Embedded Linux Conference Europe

Vienna, Austria, 2024

Bartosz Golaszewski
Linaro

About me

- Linux kernel developer for the Qualcomm Landing Team at Linaro
- 15 years of embedded linux experience
- Maintainer of the GPIO subsystem
- Author and maintainer of libgpiod
- Open-source contributor to many other projects
- Interested in complex software architecture

linaro™ is the software engine of the arm Ecosystem

Linaro empowers rapid product deployment within the dynamic arm Ecosystem.

- Our cutting-edge solutions, services and collaborative platforms facilitate the swift **development, testing, and delivery of arm-based innovations,** enabling businesses to stay ahead in today's competitive technology landscape.
- **Linaro** fosters an environment of collaboration, standardization and optimization among businesses and **open source ecosystems to accelerate the deployment of arm-based products and technologies** along with representing a pivotal role in open source discovery and adoption.
- **Automotive, Testing, Linux Kernel, Security, Cloud & Edge Computing, IoT & Embedded, AI, CI/CD, Toolchain, Virtualization**

Linaro has enabled trust, quality and collaboration since 2010

fsl,rcpm-wakeup

The problem with the GPIO character device

No state retention

- When the last file descriptor associated with a set of requested GPIO lines is closed, the state of those lines becomes undefined
- Practical example with libgpiod:
  - `gpioset foo=active`
  - `Ctrl-c`
  - `foo` is not guaranteed to be `output-high` after `gpioset` exits
- And with sysfs
  - `echo 1 > /sys/class/gpio/gpio522/value`
  - Process exited, value is retained

sysfs vs libgpiod

- the GPIO sysfs interface can be imagined as an in-kernel daemon controlling GPIOs
- if we want to provide similar functionality purely in user-space, we need a daemon process running in the background exposing a well-known interface and becoming the centralized authority controlling GPIOs
- Can we make the GPIO state persistent in the kernel?
  - No.

fsl,rcpm-wakeup

Introducing the gpio-manager

gpio-manager & gpiocli

- Has been in on-and-off development for over 2 years
- Lives under `dbus/` and `bindings/glib/` in the libgpiod tree
- What's in the package?
  - GObject bindings to libgpiod
  - gpio-manager daemon
    - implements the D-Bus API to libgpiod
  - gpiocli command line utility
  - systemd service
  - udev rules
  - example D-Bus config

Why D-Bus?

- Well-known protocol
  - Allows to export the API definitions and use whatever tools are at users' disposal (gdbus, qdbus, etc.)
- Wide selection of libraries implementing D-Bus
  - Allows to access the gpio-manager functionality from any kind of program
- In practice: present on most systems in the wild, especially ones used by makers
- Obviously: D-Bus is much slower than using the GPIOs directly over the character device so for high frequency monitoring or toggling the user may need to reconsider using it
- Will there ever be an even more lightweight solution?
  - Maybe (ಠ_ಠ)

Quick D-Bus glossary (in the context of libgpiod)

Bus names:

`io.gpiod1`

Objects:

`/io/gpiod1/chips/gpiochip0`

`/io/gpiod1/requests/request0`

Interfaces:

`io.gpiod1.Chip`

`io.gpiod1.Line`

`io.gpiod1.Request`

Quick D-Bus glossary (in the context of libgpiod)

Properties:

`io.gpiod1.Line.Offset`

`io.gpiod1.Chip.Label`

Methods:

`io.gpiod1.Chip.RequestLines()`

`io.gpiod1.Request.SetValues()`

Signals:

`io.gpiod1.Line.EdgeEvent`

D-Bus API in action

```
$ qdbus --system io.gpiod1
/
/io
/io/gpiod1
/io/gpiod1/chips
/io/gpiod1/chips/gpiochip2
/io/gpiod1/chips/gpiochip2/line5
/io/gpiod1/chips/gpiochip2/line4
...
/io/gpiod1/requests
/io/gpiod1/requests/request0
```

D-Bus API in action

```
$ qdbus --system io.gpiod1 /io/gpiod1/chips/gpiochip0 io.gpiod1.Chip.Label
INT34C6:00

$ qdbus --system io.gpiod1 /io/gpiod1/chips/gpiochip1/line1 io.gpiod1.Line.Name
foo

$ qdbus --system io.gpiod1 /io/gpiod1/requests/request0 io.gpiod1.Request.Release
```

Examples

```
# Wait for gpio-manager to come up
$ gpiocli wait

# Wait for a specific GPIO chip to appear in the system
$ gpiocli wait --chip="INT34C6:00"

# Wait with a timeout
$ gpiocli wait --timeout=1s
```

Examples

```
# Detect chips in the system.
$ gpiocli detect
gpiochip0 [INT34C6:00] (463 lines)
gpiochip1 [gpio-sim.0:node0] (16 lines)
gpiochip2 [gpio-sim.0:node1] (8 lines)

# Detect a specific chip
$ gpiocli detect gpiochip0
gpiochip0 [INT34C6:00] (463 lines)
```

Examples

```
# Display information about a line
$ gpiocli info foo
gpiochip1 1: "foo" [used,consumer="gpio-manager",managed="request0",output,push-pull]
```

Examples

```
# Request a set of lines. Note that gpiocli exits immediately
# but the state of the lines is retained because it's the
# gpio-manager that requested them.
$ gpiocli request --output foo=active
request0

# List active requests
$ gpiocli requests
request0 (gpiochip1) Offsets: [1]
```

Examples

```
# We can now change the value of the line.
$ gpiocli set foo=inactive

# Note that gpiocli exited but the value is retained (hooray!)

# Read it.
$ gpiocli get foo
"foo"=inactive
```

Examples

```
# We can even reconfigure it to input and enable edge-detection.
$ gpiocli reconfigure --input --both-edges request0

# And wait for edge events.
$ gpiocli monitor cos
21763952894920 rising  "foo"

# And finally release the request.
$ gpiocli release request0
```

Examples

```
# Find the chip and offset for a named line
$ gpiocli find foo
gpiochip1 1
```

Examples

```
# Monitor state changes of a line
gpiocli notify foo
gpiochip1 - 1 ("foo"): [output,push-pull]

# Line is requested

gpiochip1 - 1 ("foo"): [request=>request0]
gpiochip1 - 1 ("foo"): [managed=>True]
gpiochip1 - 1 ("foo"): [direction=>input]
gpiochip1 - 1 ("foo"): [consumer=>"gpio-manager"]
gpiochip1 - 1 ("foo"): [used=>True]
```

Distro integration

- Yocto recipe ready for submission once v2.2 is out
- systemd service file included with some sane sandboxing rules
- udev rules included for putting GPIO character devices in the `gpio` group
- Basic D-Bus permissions
  - root can do anything
  - anyone can inspect GPIO chips and its properties
  - `gpio-manager` user can own the `io.gpiod1` name on the system bus
  - members of the `gpio` group can request and manipulate GPIO lines

Where are we at?

- Initial version of gpio-manager and co. is now in libgpiod's master branch
- About to tag v2.2-rc1
- Reviews and testing are much needed!
- libgpiod v2.2 will contain D-Bus support

Q & A

Thank You!

Visit linaro.org

Contact me at:
bartosz.golaszewski@linaro.org